Connectivity Properties of Mainline BitTorrent DHT Nodes

Raul Jimenez Flutra Osmani Björn Knutsson

Royal Institute of Technology (KTH)
ICT/TSLAB
Stockholm, Sweden
{rauljc, flutrao, bkn}@kth.se

Abstract

The birth and evolution of Peer-to-Peer (P2P) protocols have, for the most part, been about peer discovery. Napster, one of the first P2P protocols, was basically FTP/HTTP plus a way of finding hosts willing to send you the file. Since then, both the transfer and peer discovery mechanisms have improved, but only recently have we seen a real push to completely decentralized peer discovery to increase scalability and resilience.

Most such efforts are based on Distributed Hash Tables (DHTs), with Kademlia being a popular choice of DHT implementation. While sound in theory, and performing well in simulators and testbeds, the real-world performance often falls short of expectations.

Our hypothesis is that the connectivity artifacts caused by guarded hosts (i.e., hosts behind firewalls and NATs) are the major cause for such poor performance.

In this paper, the first steps towards testing this hypothesis are developed. First, we present a taxonomy of connectivity properties which will become the language used to accurately describe connectivity artifacts. Second, based on experiments "in the wild", we analyze the connectivity properties of over 3 million hosts. Finally, we match those properties to guarded host behavior and identify the potential effects on the DHT.

1 Introduction

The BitTorrent protocol [6] is widely used in Peer-to-Peer (P2P) file sharing applications. Millions of users[1] collaborate in the distribution of digital content every day. As traditional broadcasters transition to Internet distribution, we can expect this number to increase significantly, which raises some concerns about the scalability and resilience of the technology.

Our work is part of the P2P-Next[1] project, which is supported by many partners including the EBU[2] who claims to have more than 650 million viewers weekly. In the face of such load, scalability and resilience become vital components of the underlying technology.

In BitTorrent, content is distributed in terms of objects, consisting of one or more files, and these objects are described by a *torrent*-file. The clients (*peers*) participating in the distribution of one such object form a *swarm*.

A swarm is coordinated by a *tracker*, which keeps track of every peer in the swarm. In order to join a swarm, a peer must contact the tracker, which registers the new peer and returns a list of other peers. The peer then contacts the peers in the swarm and trades pieces of data with them.

The original BitTorrent design used centralized trackers, but to improve scalability and resilience, distributed trackers have been deployed and currently exist in two flavors: *Mainline DHT* and *Azureus DHT*. Both of them are based on Kademlia[11], a distributed hash table (DHT). Kademlia is also the basis of *Kad*[17], used by the competing P2P application eMule[3].

Kademlia's properties and performance have been thoroughly analyzed theoretically as well as in lab settings. Kademlia's simplicity is one of its strengths, making theoretical analysis simpler than that of other DHTs. Furthermore, simulations such as [9] show that Kademlia is robust in the face of *churn*[4].

When we analyze previous measurements on three Kademlia–based DHTs, we find that Kad, eMule's implementation of Kademlia, has demonstrated good performance [17], while the two Kademlia-based BitTorrent DHTs (Mainline DHT and Azureus DHT) show very poor performance [7]. While lookups are performed within 5

[1] The Pirate Bay alone tracks more than 20 million peers at any given time.

[2] European Broadcasting Union

[3] http://www.emule-project.net/ (last accessed April 2009)

[4] Defined in Section 3

seconds 90% of the time in Emule's Kad, the median lookup time is around a minute in both BitTorrent DHTs.

One of the main differences between Kad and the other two implementations is how they manage nodes running behind NAT or firewall devices. Kad attempts to exclude such nodes from the DHT. On the other hand, neither of the BitTorrent's DHT implementations have such mechanisms.

Evidence suggests that some connectivity artifacts on deployed networks were not foreseen by DHT designers. For instance, Freedman et al. [8] show how non-transitivity in PlanetLab degrades DHT's performance (including Kademlia). These connectivity artifacts exist on the Internet as well, as our experiments will show.

Guarded hosts, hosts behind NATs and firewalls [19], are well-known in the peer-to-peer community for causing connectivity issues. Different devices and configurations produce different connectivity artifacts, including non-transitivity.

This evidence leads us to believe that DHT implementations which consider and counteract guarded hosts' effects are expected to perform better than those that do not.

The ultimate test of this hypothesis would be checking whether guarded host's connectivity artifacts significantly affect Kademlia's lookup performance. In order to do this, we need to understand the characteristics of these connectivity artifacts and their potential effects on the DHT routing. Then, we would be able to carry out an experiment looking for these effects on the actual lookups.

In this paper, we focus on the definition and analysis of these connectivity properties. We also underlay the potential effects on the DHT performance. Although we do not attempt to test whether guarded hosts actually degrade lookup performance, an outline of the future work is provided in Section 6.

Mainline DHT, used for BitTorrent peer discovery, was integrated into Tribler[13] —the integral component of the P2P-Next project. The ultimate goal of this ongoing research is to adapt the Mainline DHT to the non-ideal Internet environment, while keeping backward compatibility with the millions of nodes already deployed. Thus, we focus our experiments on the Mainline BitTorrent DHT nodes.

We model nodes' connectivity according to three properties: *reciprocity*, *transitivity*, and *persistence*. This taxonomy in itself is one of the contributions of this paper, since it provides the language needed to reason about and specify the connectivity assumptions made by DHT designers and deployers. For every property, we discuss the possible cause and its potential effects on Kademlia.

In our experiments, the connectivity properties of more than 3 million DHT nodes are studied. We find that most of the connectivity patterns observed correlate to common NAT and firewall configurations.

The following section provides an overview of Kademlia and guarded hosts. In Section 3 the potential effects of the connectivity artifacts are discussed. We present our experiment in Section 4, discuss the results in Section 5, outline the future work in Section 6 and conclude in Section 7.

2 Background

In this section we give the background needed to understand the experiments and the results. We provide basic information regarding Kademlia's routing table management and its lookup routing algorithm. In the second part, we overview the generic behavior found on most common configurations of NATs and firewalls.

2.1 Kademlia

Kademlia[11] is a DHT design which has been widely deployed in BitTorrent and other file sharing applications. When used as a BitTorrent distributed tracker, Kademlia's *objectIDs* are torrent identifiers and *values* are lists of peers in the torrent's swarm.

Each node participating in Kademlia obtains a *nodeID*, whereas each object has an *objectID*. Both identifiers consist of a 160-bit string. The value associated to a given objectID is stored on nodes whose nodeIDs are closest to the objectID, where closeness is determined by performing a XOR bitwise operation on the nodeIDs and objectID strings.

Every node maintains a routing table. The routing table is organized in *k-buckets*, each covering a certain region of the 160-bit key space. Each k-bucket contains up to k nodes, which share some common prefix of their identifiers. New nodes are discovered opportunistically and inserted into the appropriate buckets as a side-effect of incoming queries and outgoing lookup messages.

Kademlia makes use of iterative routing to locate the value associated to the objectID, which is stored on the nodes whose nodeIDs are closest to the objectID. The node performing the lookup queries nodes in its routing table — those whose nodeIDs are closest to the objectID. Each of those nodes returns a list of nodes whose nodeIDs are closer to the objectID. The node continues to query newly discovered nodes until the result returned is the value associated to the objectID. This value, when using Kademlia as a BitTorrent tracker, is a list of peers.

2.2 Guarded Hosts

NATs and firewalls are important components of the network infrastructure and are likely to continue to be deployed. According to a recent paper [12], two thirds of all peers are behind NATs or firewalls in open BitTorrent communities. Despite the fact that different firewalls and NATs

can have different configurations, the most common types are overviewed in this subsection.

Note that we just focus on UDP connectivity since the Mainline BitTorrent DHT uses UDP as transport protocol.

2.2.1 Firewalls

A guarded host located behind a firewall is able to send outgoing packets but may be unable to receive incoming packets. Though several firewall configurations are deployed, in this paper, we consider the simplest case where outgoing packets are forwarded but the incoming packets are dropped. In such a scenario, the connectivity is *non-reciprocal*, and the internal host is able to send but not receive any packet.

2.2.2 NAT Behavior

NAT behavior is more complex. A host behind a NAT is able to send packets to hosts on the other side of the NAT. The NAT device, in turn, keeps track of the packets sent by the internal host in its table, in the form of entries that expire within a certain timeout. When the external host replies, the NAT box checks the reply against its address translation table, before routing the reply back to the internal host. For as long as the entry remains in the NAT table, the two hosts are able to communicate, and the communication, according to our property definitions in Section 3, is said to be *persistent*.

The entries in the NAT table are either removed when they timeout or when new entries replace the old ones. Since the DHT nodes contact many other nodes, it is expected that NAT tables can fill up rather quickly.

2.2.3 NAT Timeouts

Recent measurements of NAT/firewall characteristics in the Tribler system[5] reveal that the average NAT timeout value is 2 minutes for more than 60% of the NATed hosts studied. Moreover, the IETF RFC 4787 [3] specifies that a NAT UDP entry should not expire in less than two minutes; it also recommends a default value of 5 minutes or more for each entry. However, since NAT behavior is not really standardized, applications must be extremely conservative, in order to cope with the large variation of (observed) behaviors.

When the entries are removed from the table, the external hosts are unable to reach the internal host, since NAT boxes discard all incoming packets for which they find no match in their table entries. From the perspective of an external host, the internal host is no longer reachable, while in fact, the internal host continues to listen behind the NAT box. Consequently, the size-limited tables or short timeouts of NAT devices may break *persistence*.

[5]https://www.tribler.org/trac/wiki/NATMeasurements (last accessed June 2009)

2.2.4 NAT Configuration

Usually, the NAT (or firewall) device behind which the node is sitting is under the control of the user. Most of the issues created by them can be resolved, or at least mitigated to a large extent, by proper configuration. In many cases, this is as simple as enabling Universal Plug and Play-support[2] (UPnP) in the NAT-box, and have the DHT implement a UPnP-client to correctly setup forwarding.

Alternatively, the DHT application could provide the information needed by the user to manually configure the NAT to forward UDP traffic.

2.2.5 STUN

When participating in the DHT, a node will keep a routing table with pointers to other DHT nodes. Additionally, it will be a tracker for a small number of BitTorrent objects. The role of a node in DHT is to receive queries from other nodes —either updating routing information or performing DHT lookups. In either case, this is a very light weight computational operation.

Session Traversal Utilities for NAT[16] (STUN) may initially seem like an option for dealing with NAT traversal. STUN is certainly possible to implement, and perfectly reasonable for setting up VoIP streams and other long-term communications. However, unlike VoIP streams, DHT messages are very short-term communications (usually a single query/response) and the number of connections to different nodes is high (commonly a few hundred).

For the DHT as a whole, we argue that the cost of using STUN to reach an otherwise unreachable node exceeds the benefit gained by having that node participate in the DHT.

3 Dissecting Churn — Property Definitions

In a DHT, any node can join or leave the DHT at any moment. *Churn* is measured as the number of nodes joining and leaving the DHT during a given period of time, and is thus an indicator of how dynamic a DHT network is.

Since each node needs to keep its routing table updated and accurate, a maintenance overhead is associated with churn. That is why counteracting churn is so important when designing and deploying a DHT.

Much research has been done on DHT performance in presence of churn [15, 18]. Our hypothesis, however, is that a large fraction of the reported churn in deployed DHTs is caused by connectivity artifacts. Unlike real churn, this *apparent churn* follows certain patterns which may be used to identify it and, potentially, eliminate it.

Although we have not attempted to perform similar experiments on other Kademlia-based implementations, we expect that our findings hold for all implementations which

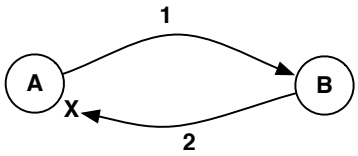

Figure 1. Non-reciprocal connectivity

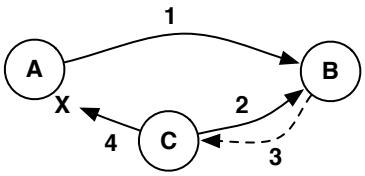

Figure 2. Non-transitive connectivity

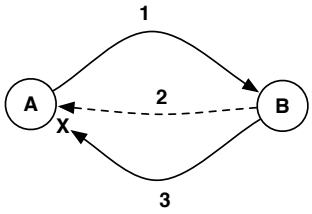

Figure 3. Non-persistent connectivity

do not have explicit mechanisms in place to mitigate guarded nodes' effects on the DHT.

In the next subsections, we define the three connectivity properties we have identified, along with the percentage of DHT nodes exhibiting them in the measurements we have performed. We also examine plausible reasons why a large fraction of nodes are missing one or more of these properties, and how this will impact the performance of the DHT.

Throughout the subsections, the numbers accompanying the protocol descriptions refer to the message exchange order and match the numbers in the corresponding figures.

3.1 Non-reciprocal Connectivity May Create Apparent Churn

On the open Internet, it is assumed that if node A can establish a connection to node B, then node B can establish a connection to node A, i.e., connectivity is *reciprocal*. Our experiments, however, reveal that just **80%** of the nodes exhibit reciprocal connectivity. Firewalls and NATs which forward outgoing, but drop incoming, packets are the likely cause.

Figure 1 depicts the non-reciprocity of the connectivity between A and B. In this scenario, A is the node behind a firewall and the one to initiate the connection with B (1). B assumes the connectivity to be reciprocal and thus inserts A in its routing table.

After a while, when refreshing the buckets in the routing table, node B finds that A no longer replies to its queries. After several failed attempts to reach A (2), B regards A as unreachable and therefore removes it from the routing table.

After being removed from the routing table, A may send a new query to B. As before, B would consider A a good candidate for its routing table and therefore start the process over again.

3.2 Non-transitive Connectivity May Break Lookup Routes

On the Internet, there is a general assumption of *transitivity*, meaning that if node A can reach node B, then any node that can reach B will also be able to reach A. NATs and firewalls break this assumption, and in fact, less than **40%** of the nodes analyzed have transitive connectivity.

Figure 2 illustrates the case, where node A, which is located behind a NAT, causes non-transitive connectivity. When node A sends a query to B (1), B replies back and adds A to its routing table. Later, when node C is performing a lookup, it queries B (2) and B replies with a reference node from its routing table (3), which in this case is A. On the next lookup step, node C sends a query to A (4), but receives no reply. If the connectivity were transitive, C would have been able to reach A, but in this case, C will eventually wait for a timeout —confirming that A is unreachable— and attempt to use an alternate node. Or, formally expressed: C can reach B (2), B can reach A (reply to 1), but C is not able to reach A.

DHTs employing iterative routing, such as Kademlia, are affected by non-transitive connectivity. Concretely, non-transitive connectivity breaks lookup routes.

3.3 Non-persistent Connectivity May Create Apparent Churn

Persistence is a more vague concept. We say that A node exhibits persistent connectivity if it can be reached all the way from the moment it joins until it leaves the DHT.

As explained in Section 2.2, NATs are known to cause ephemeral connectivity. In Figure 3, the connectivity between node A and B is a non-persistent one, where A is located behind a NAT.

Immediately after node A sends a message to B (1), node A is able to receive messages from B (2). Assuming that A does not leave the DHT, B should be able to reach A at any given time. In this case, however, the connection breaks down and node B is unable to reach A (3).

This behavior could be explained in terms of generic NAT behavior, as described in Section 2.2. The NAT enables connectivity between A and B, but only for the period

of time when the entry in the NAT table is valid, after which the connection is effectively broken.

In our experiments, slightly less than **44%** of the nodes were reachable during, at least, a five minute window. Please note that some of the unreachable nodes could be legitimately unreachable, i.e., due to actually leaving. Similarly, some of the reachable nodes may have been reachable only because of communication from other nodes "refreshed" the relevant NAT table entry.

4 Experiment Description

In this experiment, DHT nodes' reachability is analyzed from three different vantage points. Every time a node sends a query to one of our instrumented DHT nodes, queries are sent from (1) the same IP and port number, (2) same IP but different port number, and (3) a different IP.[6] The process is repeated after a period of 5 minutes.

The pieces of software developed are described in the following subsections. Both source code and result logs are available online.[7]

The setup consists of one PC running Ubuntu GNU/Linux. This computer is assigned 17 IP addresses which are managed through virtual interfaces. *remotechecker* is associated with one of the virtual interfaces. While an *instrumented DHT node* and a *localchecker* are associated with each of the rest of virtual interfaces.

Our DHT nodes' identifiers are chosen in a way that the first four bits are different from each other. This "spreads out" our nodes in the identifier space. The aim of this configuration is to broaden the DHT identifier space coverage in order to discover as many nodes as possible.

Figure 4 illustrates the reachability analysis process. Numbers in the arrows indicate chronological order and are referenced throughout the following subsections.

4.1 Rechability Checker

We have developed a piece of software called Reachability Checker (*RChecker*). Rchecker checks and logs reachability information regarding a given DHT node.

Nodes are identified by their IP address and nodeID. Nodes with different nodeIDs and same IP could be different nodes running on the same host or on different hosts behind a common NAT. Nodes with the same nodeID and different IP should not exist on the DHT. The latter nodes exist, albeit in small numbers, and are considered in the results. When two queries are received from the same IP and nodeID but different port, they are considered as coming from the same node, and just the first instance is considered.

[6]The reference point is our modified DHT node (IP address and port).

[7]http://tslab.ssvl.kth.se/raul/p2p09/

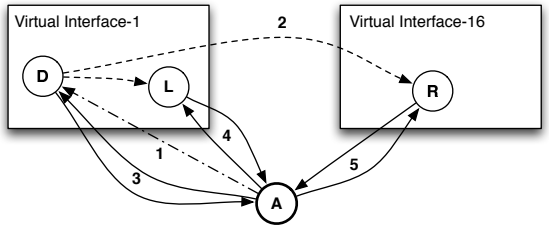

Figure 4. Experiment setup

Subsequent queries from nodes that were already checked are discarded.

Every time a node is to be checked, RChecker sends a burst of queries to the node. Queries are sent every 5 seconds, up to 5 times. Once RChecker receives a reply, a *reachable* status is recorded and no more queries are sent. This multiple querying should avoid recording reachable nodes as unreachable due to temporary network conditions.

If no reply is received within one minute, an *unreachable* status is recorded. Note that most of the BitTorrent Kademlia implementations have a 20 seconds timeout. Some have argued that 20 seconds is already too long and actually harms lookup performance [7]. In this experiment, however, we have chosen such a long timeout because we want to be able to detect network connectivity; even when the round trip time is longer than a DHT implementation's timeout would be.

A second burst, identical to the one described above, is sent 5 minutes later.

4.2 Instrumented DHT Node

We have instrumented Tribler's implementation of Kademlia[8]. The original code is modified to call RChecker as needed. Additionally, the socket used by Kademlia is passed to RChecker, so the queries are sent using the same source IP and port.

Everytime a query is received (1) and the node has not been already checked, the node's information (IP, port, ID) is sent to localchecker and remotechecker (2). Then, RChecker is called in order to check the node's reachability using the same IP and port as the DHT node (3).

4.3 Localchecker

Every localchecker is listening to the instrumented DHT node sharing the same virtual interface —i.e., both have the same IP address. Every time localchecker receives information from the instrumented DHT node (2), it calls RChecker to check the node's reachability from the same IP address as the DHT node but different port (4).

[8]http://svn.tribler.org/khashmir/ (last accessed June 2009)

Table 1. Experiment results and possible causes

Pattern	Nodes (%)	Possible cause(s)
UUU-UUU	10.6	Firewall
RUU-UUU	**31.3**	Port restricted cone
RUU-RUU	2.8	and symmetric NAT
RRU-UUU	0.8	Restricted cone NAT
RRU-RRU	2.0	
RRR-UUU	2.7	Full cone NAT and real churn
RRR-RRR	**35.5**	Open Internet
UUU-RRR	7.6	Behavior not matched
RRU-RRR	1.7	
Other	5	Rest of the cases

4.4 Remotechecker

The single remotechecker listens to every instrumented DHT node. Every time remotechecker receives information from the DHT node (2), it calls RChecker to check the node's reachability from an IP address which is different from the one used by the instrumented DHT node (5).

5 Experiment Results

During 24 hours of running the experiment, 3,683,524 unique nodes were observed. Table 1 shows the observed connectivity patterns along with the NAT or firewall types, matching the pattern and the percentage of nodes.

The notation used throughout this section is `XXX-XXX`, where the `X` can be either `R` (reachable) or `U` (unreachable). The connectivity fingerprint of each checked node can be represented by this 6-character string.

The first character accounts for the reachability of the node from the instrumented DHT node (same IP and same port). The second character represents the reachability of the node from *localchecker* (same IP but different port). Likewise, the third one indicates wheather the node is reachable or not from the *remotechecker* (different IP).

The last three characters follow the same structure, however, they represent node's connectivity after a 5 minute period.

5.1 Analysis

More than **10%** of the nodes are globally unreachable (`UUU-UUU`). They are able to send messages — our modified DHT node received at least one query from them. They are, however, unable to receive messages from any of our vantage points. This connectivity pattern matches the firewall behavior, configured to let outgoing messages through but drop incoming messages.

A large percentage of the nodes in the DHT population are only partially reachable. Typically, they can be reached only under certain circumstances. We argue that NATs are the main cause of this partial reachability of nodes.

As explained in Section 2.2, NAT devices have a timeout parameter which make stale entries expire after a given period of time. In table 1, NAT types have two associated observed patterns. The former matches the case when the NAT entry expires within 5 minutes, therefore, the node is unreachable the second time it is checked. In the latter, the NAT timeout is longer than 5 minutes. Notice that a *full cone* NAT, whose entry has not expired, matches the open internet behavior.

More than **34%** of the nodes are reachable from our modified DHT node, but neither from *localchecker* nor *remotechecker* (`RUU-RUU` and `RUU-UUU`). This behavior matches *port restricted cone* and *symmetric* NAT types. These NAT types register outgoing connections that are initiated by an internal host. An incoming packet is only forwarded to the internal host if both the IP and port of the external host match the NAT's entry. Packets coming from the same IP but different port (*localchecker*) or a different IP (*remotechecker*) are discarded by the NAT device.

About **3%** of the nodes are reachable from our modified DHT node and the *localchecker* but not from the *remotechecker* (`RRU-UUU` and `RRU-RRU`). The plausible explanation is that the node is behind a *restricted cone* NAT, in which case, the incoming packets are forwarded only when the IP address of the external host matches the NAT's entry. Therefore, packets coming from our modified DHT node and the *localchecker* (same IP) are received by the analyzed node, while those from the *remotechecker* are dropped.

Less than **3%** of the nodes in the DHT are reachable from the instrumented DHT, the *localchecker* and the *remotechecker* during the first time when testing their connectivity (`RRR-UUU`). However, the nodes are globally unreachable when checked after a period of 5 minutes. The probable cause of this pattern is a full cone NAT, whose corresponding entries in the NAT table have expired within the testing period. This case will be further discussed in this section.

Approximately **35.5%** of the DHT nodes are globally reachable. They are reachable from all of our vantage points before, as well as, after the 5 minute waiting period.

Finally, we show two patterns that do not match any of the expected behaviors but represent more than 1% each. They are `UUU-RRR` (7.6%) and `RRU-RRR` (1.7%). These cases remain open for further research.

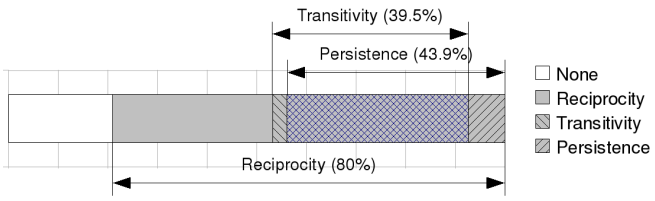

Figure 5. Properties Chart

5.2 Correlation between Transitivity and Persistence

Figure 5 depicts transitivity (`RXR-XXX`) and persistence (`RXX-XXX`) as subsets of reciprocity (`RXX-XXX`). We also notice the significant overlap between transitivity and persistence. This overlap was indeed expected since NAT devices commonly cause both non-transitivity and non-persistence, as previously discussed. Furthermore, some of the cases exhibiting persistence and non-transitivity might be due to long NAT table timeouts or casual messages – "refreshing" the right NAT table entry.

Based on the above observations, we can formulate the heuristic that if a node's connectivity is known to be transitive, it is very likely to be persistent as well, and vice versa. By applying this heuristic, we may be able to use a less expensive method of testing connectivity, without a significant loss of accuracy.

5.3 Apparent & Real Churn

Since we identify the nodes' properties from an outsider's point of view, we do not know what connectivity properties a given node actually has. This fact complicates the task of differentiating apparent churn from nodes effectively leaving the DHT (i.e., real churn).

We can certainly say that any node which replies to one or more of our venture points after the 5 minute period, has not left the DHT. Therefore, connectivity patterns in the `XXX-UUU` category might be caused by nodes actually leaving the DHT. This category accounts for 45.6% of the nodes.

The `UUU-UUU` pattern (10.6%) belongs to this category. These nodes fail to reply to us immediately after they have sent us a query[9]. Since the time is so short (a UDP round trip) we can assume that very few nodes, if any, would have left the DHT within such extremely short period. Instead, we argue that this is apparent churn caused by firewalls.

Another interesting pattern is `RRR-UUU` (2.7%). This pattern may be caused by *full cone* NAT which forwards the traffic to the internal host regardless of the source's IP address, but the NAT entry would expire within the 5-minutes window. However, according to our observations, DHT nodes constantly receive and send messages which refresh the NAT entries, thus making the connections effectively open, given a long enough NAT timeout. This fact makes us believe that a good part of these cases corresponds to real churn —i.e., nodes in the open Internet which have left the DHT within the 5-minutes window.

The case which accounts for most of the nodes in the `XXX-UUU` category is `RUU-UUU` (31.3%). The fact that these nodes have limited connectivity in the first place makes them unfit to carry out DHT tasks. Therefore, DHT implementations that avoid adding nodes with limited connectivity into the routing tables, will most likely not experience the churn issues caused by NATs, notoriously reducing DHT's churn.

[9]The query triggers the reachability check.

6 Related and Future Work

6.1 Dealing with Limited Connectivity Nodes

As previously stated, our hypothesis is that limited connectivity often is a result of an improperly configured NAT/firewall. The very first step towards dealing with limited connectivity should thus be to properly document the requirements of the client, and to make it easy to configure and test port forwarding in the NAT/firewall for the client.

Still, the DHT must be able to cope with the problems limited connectivity nodes pose, and we have seen in Emule's Kad [17] that fairly simple modifications to existing DHT implementations can go a long way towards mitigating the effects of limited connectivity.

Many of the proposals and perfomed simulations have mainly tried to mitigate the negative effects and improve the overall performance of the DHT, but none has addressed the underlying problem. Moreover, their benefits come at the cost of other performance factors, mainly bandwidth consumption. We find examples of such improvements in [7, 4, 9]:

- Check node's reachability before adding it to the routing table.
- Reduce timeout value or implement adaptive timeouts.
- Increase lookup parallelism.
- Increase the refresh rate, such that dead nodes are discovered earlier.
- Implement an "extended table" or bigger size routing tables, such that the probability of having fresh entries in the routing table increases.

- Maintain small and fresh routing tables, by removing neighbors whose estimated probability of being alive is below some calculated threshold [10].

The above parameter fine-tunings should be considered in the widely-deployed Kademlia DHT, where millions of users simultaneously participate today and tens of millions of users may participate in the near future. The increase in user participation implies larger routing tables, and a potentially exponential growth in the number of maintenance messages. The proposed tweaks requiring additional messages would exacerbate this growth, and may prove an obstacle to DHT scalability. Tweaks that only require local resources, i.e., memory and processing, are much more likely to scale, and will benefit from Moore's Law.

Another approach is to try to determine the specific properties of hosts before adding them to the routing table, see the discussion in Section 3. Rhea proposes a set of measurements in order to counteract the effects of non-transitive connectivity on OpenDHT [14].

As our experiment demonstrates, the connectivity properties essential to a DHT of a given node can be determined. Reciprocity is easily detected by sending a single query, while checking transitivity is more complex to detect. The strategy used in our experiments relied on multiple IP addresses being available to the test host, but we can't expect a normal DHT node to have access to more than a single IP address.

While one DHT node could use another node as *remotechecker*, letting it relay queries and report the reachability status, this opens a whole new can of worms. For example, this mechanism could be exploited for DDoS[10] attacks.

Nevertheless, *localchecker* can be easily implemented and deployed without the need of several IP addresses per host or additional trust. In fact, *localchecker* is able to correctly identify most of the non-transitive connectivity cases. Based on our results, only 4.6% of them are detected by *remotechecker* but not by *localchecker*. Thus, if only the local mechanisms were to be applied, we would still vastly improve the quality of nodes in the routing table. Furthermore, we would avoid introducing excessive complexity and security vulnerabilities.

An additional mechanism that would improve the quality of the routing table content is to quarantine new nodes before adding them to the routing table. This gives enough time to perform a second reachability check in order to determine whether the candidate node's connectivity is persistent, similarly to the approach used in our experiment.

As seen previously in Figure 5, transitivity and persistence are correlated but do not completely overlap. Therefore, either *localchecker* or quarantine alone would identify most of the limited connectivity nodes, but a combination of both would correctly classify the vast majority of nodes, thus increasing the detection effectiveness.

The mechanisms described above can be combined with a policy that is consistent with our discussion in 2.2.5 — where guarded nodes are not allowed to join the DHT. In Kad, however, the node will instead find a DHT node to use as a proxy. While this approach has been used in a fairly large deployment, it moves load and responsibility to nodes already in the DHT, thus adding complexity by requiring a separate proxy mechanism/protocol.

A policy similar to the one used in StealthDHT[5] might be more appropriate. According to StealthDHT, nodes participating in the DHT are separated in two categories: *service nodes* and *stealth nodes*. Service nodes perform routing and value storage tasks, while stealth nodes are not involved in any active task but are able to maintain their own routing tables and perform lookups.

We would like to take a further step and add conceptual as well as practical separation. Nodes which are able to, will be part of the DHT and act as a *service node*, handling routing and storage of values. Nodes with limited connectivity will only be clients. As such, they will perform their own lookups using DHT nodes, and they may even cache information locally, but they will never be contacted by other nodes. Finally, notice that, due to Kademlia's iterative routing, service nodes only need to reply to simple queries, while DHT clients can initiate and keep track of the lookup's state on their own.

6.2 Future Work

We have argued that the large percentage of DHT nodes having limited connectivity has repercussions on the DHT performance. They become passive participants of the routing tables, only causing delays and stale entries.

At the most basic level, Emule's Kad implementation tries to detect nodes that don't reply to queries, and completely excludes them from the DHT. However, since we can find no references to how or why this was done, we are unable to determine whether this was an *ad hoc* solution, or it was the result of careful design based on a study similar to ours.

In this paper, we have studied the connectivity properties of nodes by deploying a set of DHT nodes and studying the properties of nodes which exchange messages with them. However, the fact that guarded hosts exist, and are active in the DHT, is not a problem per se. It is only when routing tables are effectively poisoned that lookup performance declines. A logical next step would thus be to take inventory of the routing tables of DHT nodes "in the wild", and find out to what extent guarded nodes actually end up in routing tables.

[10] DDoS stands for Distributed denial of service.

Furthermore, our ultimate goal is to use the knowledge we have gained from this research to repair and improve the DHT performance. As dissussed in the previous subsection, that would include designing mechanisms that identify/remove limited connectivity nodes from the routing table, and prevent such nodes from being added in the first place.

Finally, it could be instructive to compare the "pollution rate"[11] in the routing tables of different DHTs, such as the Mainline and Azureus DHT, as well as eMule's Kad.

7 Conclusion

In this paper, we have defined a set of properties which provides the language needed to spell out the assumptions made by DHT designers and deployers. These properties were not explicitly considered in the original Kademlia design. Instead, their effects were only discovered when DHTs were deployed and faced with the non-ideal connectivity artifacts in the real world.

We have studied over 3 million BitTorrent Mainline DHT nodes' connectivity according to these properties. The results point to the generalized presence of NAT and firewall devices causing connectivity issues in the DHT. In fact, only around one third of the nodes analyzed have "good connectivity" —i.e. reciprocal, transitive, and persistent.

Finally, we do not propose a stopgap solution for poor DHT performance. Instead, we offer the taxonomy to explicitly specify the DHT's connectivity assumptions and the toolkit to determine whether those assumptions are met. Our long-term ambition is to enable ourselves and others to design and implement DHTs where the underlying problems are addressed, instead of just tweaking parameters and adding kludges to handle the symptoms.

Acknowledgment

The research leading to these results has received funding from the Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 21617 (P2P-Next).

We would like to thank Lucia d'Acunto, Delft University, The Netherlands, for providing us with preliminary results of her experiments on NATs.

References

[1] P2P-Next Project. http://www.p2p-next.org/ (last accessed June 2009).

[2] UPnP Forum. Internet Gateway Device (IGD) V 1.0. http://www.upnp.org/ (last accessed June 2009).

[3] F. Audet and C. Jennings. Network Address Translation (NAT) Behavioral Requirements for Unicast UDP. Technical report, BCP 127, RFC 4787, January 2007.

[4] A. Binzenhfer, H. Schnabel, A. Binzenhfer, and H. Schnabel. Improving the performance and robustness of kademlia-based overlay networks, 2007.

[5] A. Brampton, A. MacQuire, I. A. Rai, N. J. P. Race, and L. Mathy. Stealth distributed hash table: a robust and flexible super-peered dht. In *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, New York, NY, USA, 2006. ACM.

[6] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, volume 6. Berkeley, CA, USA, 2003.

[7] S. A. Crosby and D. S. Wallach. An analysis of bittorrent's two kademlia-based dhts, 2007.

[8] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica. Non-transitive connectivity and dhts. In *In Proc. of the 2nd Workshop on Real Large Distributed Systems*, 2005.

[9] J. Li, J. Stribling, T. M. Gil, R. Morris, and M. F. Kaashoek. Comparing the performance of distributed hash tables under churn. In *In Proc. IPTPS*, 2004.

[10] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek. Bandwidth-efficient management of dht routing tables. In *NSDI*, 2005.

[11] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. pages 53–65, 2002.

[12] J. Mol, J. Pouwelse, D. Epema, and H. Sips. Free-Riding, Fairness, and Firewalls in P2P File-Sharing. In *Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing-Volume 00*, pages 301–310. IEEE Computer Society Washington, DC, USA, 2008.

[13] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: a social-based peer-to-peer system: Research articles. *Concurr. Comput. : Pract. Exper.*, 20(2):127–138, 2008.

[14] S. Rhea. *OpenDHT: A Public DHT Service*. PhD thesis, UNIVERSITY OF CALIFORNIA, 2005.

[15] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[16] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN-simple traversal of user datagram protocol (UDP) through network address translators (NATs). Technical report, March 2003. RFC 3489.

[17] M. Steiner, T. En-Najjary, and E. W. Biersack. A global view of kad. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 117–122, New York, NY, USA, 2007. ACM.

[18] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202, New York, NY, USA, 2006. ACM.

[19] W. Wang, H. Chang, A. Zeitoun, and S. Jamin. Characterizing guarded hosts in peer-to-peer file sharing systems. In *IEEE GLOBECOM Global Internet and Next Generation Networks Symposim*, 2004.

[11] Number of limited connectivity nodes versus the total number of nodes in the routing table.